The age of the nonprogrammer user of computing systems is at hand, bringing with it the special need of persons who are professionals in their own right to have easy ways to use a computing system. Through the programming language discussed in this paper, executives and other office personnel can perform data and word processing and communications via terminals. This language, called Office-by-Example, provides rich and powerful access to the computing system computation, data base, communication, and display facilities. Discussed and illustrated by examples are a two-dimensional screen editor, triggers, and data bases, as well as word processing, electronic mail, customized menus, and application development.

Office-by-Example: A business language that unifies data and word processing and electronic mail

by M. M. Zloof

In recent years, we have been witnessing a rapid evolution in the world of computer technology. This technology is advancing so rapidly that interactive computing in businesses is commonplace, and computers in the home are becoming a reality. Thus the age of the nonprogrammer professional, i.e., the age of end users, is emerging. These users, although professional in their own fields, have neither the time nor the motivation to learn a conventional programming language. Furthermore, the spectrum of end users is rapidly widening. Although a couple of years ago end user systems tried to address secretaries and clerks, now the range includes executives, middle managers, secretaries, engineers, clerks, and—perhaps soon—housewives.

A consequence of cheaper available hardware is that users are becoming more sophisticated in their applications and require more functions with better flexibility and ease of use. For example, users of conventional word processing systems are requesting advanced data

processing functions. To meet some of these demands data base systems are being linked to various word processors to enhance their capabilities. Naturally, the more functions a system offers, the wider the spectrum of its users, especially among the upper management executives, who are more inclined to use a system that offers easy access to remote and local data bases.

We are therefore faced with the following dilemma: power versus ease of use. As the repertoire of facilities and functions increases, it becomes more complicated to use a system with the conventional approach of preprogrammed menus. A system that offers limited functions can be interfaced by displaying a single menu to the user, but a sophisticated system with a larger domain of facilities requires many such menus, together with the ability to enter parameters at various points. Thus the complex system is not as transparent to the user as the simpler system. We are all familiar with such end users' statements as "I had to go through so many menu options to do a very simple task."

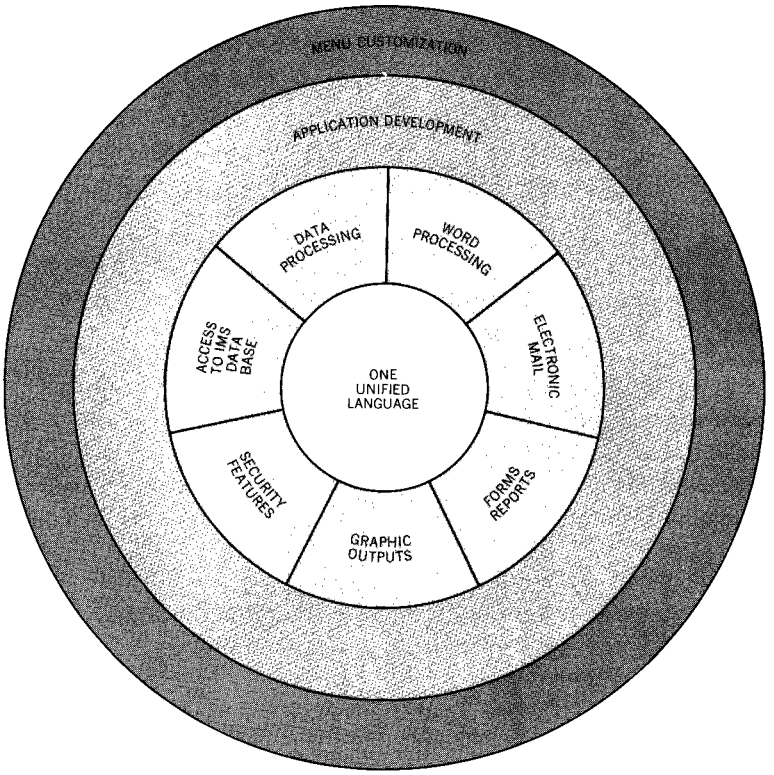
Prespecified menus have another drawback in that they lack flexibility. If a user wishes to modify a function or add a new option, he must consult the developers of the system or the programmers who maintain it. In addition, the formats of any object—be it a letter template (or skeleton), a form, a report—are all preprogrammed into the system with little or no ability for an end user to modify or define a new one.

As more and more office functions become automated through diverse products, either offered or soon to be offered in the market, the user has to learn different software manuals and languages to interface with these products. Thus a user may have to use one manual for data processing transactions, another for document handling, and yet another for audio message handling. As we shall see later, to accomplish those three functions the same file may be invoked, but its use varies with function due to the fact that the products mentioned have been developed separately by unrelated efforts with different philosophies.

What we need, therefore, is a powerful end user oriented language with which users can describe their application to the computer. This language should require minimum training, yet be powerful enough to cover a large domain of facilities.

This article describes a programming language we call Office-by-Example (OBE), which is a two-dimensional language and system that is an attempt to mimic manual procedures of business and office systems. OBE is a superset and natural extension of the Query-by-Example[1-7] data base management system, and contains features from our earlier work on a System for Business Automation.[8,9] Other research in this area is listed in References 10–18.

Figure 1 Office-by-Example facilities



QBE, a relational system,[19] is an IBM product that is used in such applications as distribution, finance, government, manufacturing, processing, construction, and utilities. It is also used in many IBM locations. At the Thomas J. Watson Research Center alone, QBE is used in about one hundred applications by over two hundred users. Reactions of various customers on the use of QBE are found in References 20 and 21; psychological studies conducted on the language, with comparison to others, show very favorable results.[22,23] OBE is a Research Division project, currently in various stages of architecture and development.

In the following sections we describe the OBE language and facilities and the components of the OBE system. We then give illustrative examples of end user programs and applications written in OBE.

The OBE language

The language for Office-by-Example (OBE) represents an attempt to combine and unify aspects of word processing (including editing and formatting), data processing, report writing, graphics, and electronic

mail. With such a language, end users (secretaries, engineers, clerks, etc.) are able to specify and store complex OBE programs, thus developing their own applications. This concept is illustrated by Figure 1. In addition, end users can set up menus (which in essence are selections of stored OBE programs) for their use or for others who are not motivated to learn details of the OBE language. An executive can specify to an assistant the menu(s) he would like to select from and the results to be displayed by the execution of the particular program selection. As an example, the executive may want a menu in order to see his mail and various summary reports (which aggregate data from a data base), together with the ability to send messages. Others may wish to see their calendars or to reserve a room automatically. In all cases, the assistant or secretary can set up an appropriate stored program based on the executive's specifications. The novelty in this approach is that menus are not preprogrammed by the system developers; rather, they are customized according to specification and set up (programmed) by end users for end users. The programming style of OBE is the same as that of QBE: direct programming within two-dimensional pictures of business objects. The user of QBE requires very little additional training to be able to use OBE.

objects and their operations

Although the fundamental data object in QBE is the table, in OBE the objects are more general, and include letters, forms, reports, charts, and graphs, as illustrated in Figure 2 (see page 282).

A fundamental concept in OBE (as in QBE) is that of *example element* variables. By utilizing these variables, users are able to program within fields of different objects. (Example elements are specified in this paper as underlined strings of characters such as X, LM, or G5.) The concept of the example element is relatively simple, yet it allows end users to perform a wide variety of operations such as the following:

- Cross-referencing between fields by entering identical example elements in two or more fields of the same or different objects.
- Formulating conditions of field values (e.g., X + Y > 50).
- Moving data from one object to another (e.g., from a data base table to a letter or a report).
- Deriving new fields.
- Locating text by using partially underlined strings of characters, examples of which are given later in the paper.
- Distributing objects to a dynamic list of destinations.

Another innovation of OBE is that end users themselves create (define) business objects on the two-dimensional display in much the same way they create these objects manually on paper. This contrasts with conventional systems in which objects are usually predefined by application programmers. Furthermore, since the QBE data base management system is its base component, users can easily extract

data from the data base and copy it into the body of the objects. Objects can also be edited and distributed through a communication subsystem to other nodes by specifying the recipient users' IDs. The distribution list can be either static (i.e., predefined, such as a list of all managers) or dynamic (i.e., variable, such as a list of all salesmen who exceeded their quotas in the previous month). The distribution list is found by the system upon issuing a query to the data base at the time of the distribution.

Another important feature of OBE is the ability to express various *trigger conditions*. When activated, these result in one or several actions. For example, one could compose a congratulatory letter to be sent to salesmen when they exceed their quotas. To check the quotas, the system evaluates the data base with a specified frequency. For each salesman whose sales exceed his quota, a trigger is activated and the congratulatory message is sent. Naturally, the data base is a key OBE component because it participates in all stages of the system: creation, editing, storage and retrieval, distribution, and trigger evaluation.

summary of the OBE facilities

The OBE language can be characterized as a nonprocedural, two-dimensional language, the facilities of which are now summarized. The syntax used here is not that of OBE, which is illustrated through the examples in the succeeding section.

Input. Input of data to the OBE system can be achieved by (a) interactively entering it through the display terminal, (b) receiving it from other nodes (other users) of the system, or (c) receiving it from remote data bases (e.g., IMS).

Data structures. OBE data structures are two-dimensional and are defined by end users. Data structures include the following:

- Two-dimensional data objects, including
  RELATIONS
  FORMS
  REPORTS
  IMS HIERARCHICAL STRUCTURES
  DOCUMENTS
  MENUS
- Two-dimensional program objects, which are a collection of various data objects and their operations.

Data types. Within fields of the OBE objects, the following data types can be declared:

CHAR
FIXED
FLOAT
DATE
TIME

Conditions. By entering constants and example elements (variables) in the appropriate fields of the various OBE structures, one can achieve the following conditions:

- Conditions on a single field.
- Conditions between fields.
- Conditions between fields of the same data base records (e.g., SAL > COMMISSION). This kind of condition is called RESTRICTION in relational algebra.
- Conditions between fields of different data base records. This kind of condition is called JOIN in relational algebra.
- Conditions between fields of different structures. For example, a field in a table must be greater than or equal to a field in a form.
- Conditions on aggregates of field values.
- Logical conditions (e.g., (SAL + COM = 50 or 60) AND W = 30).
- Integrity constraints, which are conditions that must hold to ensure the correctness of the data base.
- Authority constraints, which are conditions that protect the data base from unauthorized access and modification.

Data base query. OBE allows the user to define and query a relational data base (Query-by-Example) as well as query a hierarchical (IMS) data base. The query language is relationally complete. That is, it has the equivalent of the following relational algebra operations:

- SELECTION
- PROJECTION
- JOIN
- INTERSECTION
- UNION
- DIFFERENCE

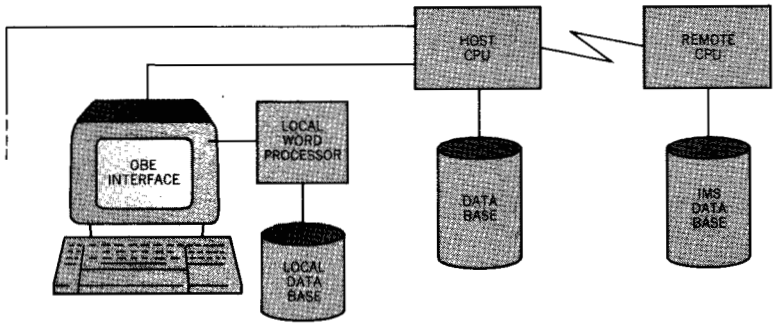
Branching. Branching in OBE is achieved by means of the *trigger* facility. When a trigger is activated, it results in one of the following actions:

- DISPLAY an object
- MODIFY an object
- SEND an object
- EXECUTE an object

These OBE facilities may be viewed as generalized DOs as in other programming languages. The ELSE clause can be achieved by taking an action when the trigger does not activate.

Nesting of programs. OBE allows the execution of programs within the scope of other programs, thus achieving the equivalent of program nesting.

Figure 3 System hardware configuration



Menu construction. OBE allows end users to set up menus, the selections of which are OBE stored programs.

Commands. OBE uses commands to execute, send, print, display, delete, or update. Some commands are entered in a special object called a *command box*.

Output. OBE allows output data to be displayed on the screen, sent to a printer, mapped to a disk, or sent to other locations.

OBE system components

In this section, an overview of the various OBE components from a system point of view is presented exclusive of the details of their application by end users. Users are introduced to the system by learning the language interface described in later sections of this paper.

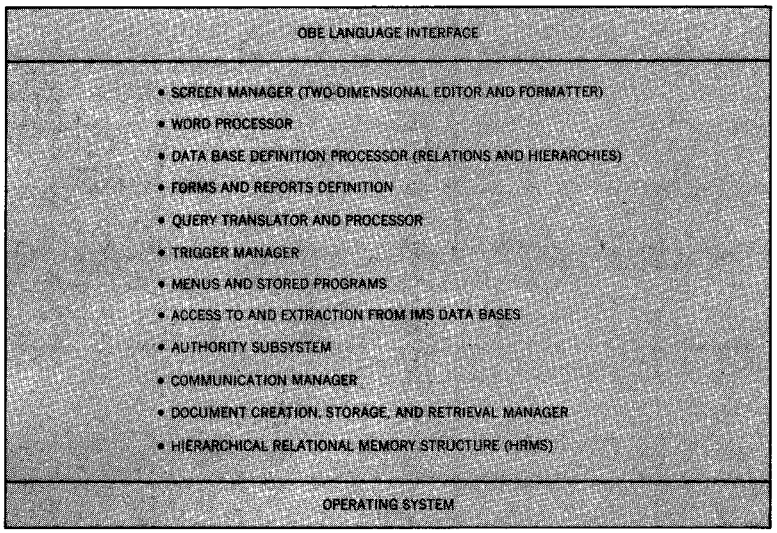
The current implementation of OBE is carried out under the VM/CMS operating system. Since, however, the language concepts and the system configuration are theoretically independent of the underlying operating system, we try not to depend on specific features of the operating system. Ideally, we are seeking an architecture whereby word-processing functions and access to personal data bases can be carried out locally at the terminal site. Data processing functions and access to remote large data bases are executed at the host, which may connect to other large main frames via a network, as illustrated in Figure 3.

The various system components are shown in Figure 4. We now describe these components in more detail.

Screen Manager—two-dimensional editor

Since OBE is a two-dimensional language, the Screen Manager (SM) supports and manipulates two-dimensional objects very flexibly. OBE is written to support various terminals (IBM 3277, 3278, and the 3279

Figure 4 Office-by-Example system components



color terminal), and with little effort can support an All Points Addressable (APA) terminal. (The screens pictured in this paper were produced using the IBM color display.)

The Screen Manager supports the following two-dimensional structures.

Multiple windows (view ports): A window (or view port in graphics terminology) is a box that can be displayed on a screen by pressing an appropriate function key. Windows and other objects are displayed relative to the position of the cursor (with the top left corner of the generated object at the cursor position). Each object is displayed as though it is in a different depth plane. Thus objects can move over the top of and relative to one another. Each window in OBE constitutes a PROGRAM OBJECT. Therefore, two windows displayed simultaneously on a screen can run two independent programs.

Multiple objects within a single window: Various business objects can be displayed, moved, and scrolled within a single window. A box is used to compose a letter, a table to define or to query a relational data base, an x-y coordinate to define a graph, and so forth. We distinguish between two types of objects, primitive and derived. *Primitive objects* are prestored objects that can be displayed by pressing appropriate function keys, such as the skeleton of a box or the skeleton of a table. *Derived objects* are more complicated in that they are created by end users and stored in the system for later use. Examples are invoice form, memorandum, letter, or report skeletons. Note that these derived objects are not preprogrammed into the system by professional programmers (as it is done traditionally), but

are defined by end users mimicking the way these objects appear on paper. (An example of how an end user defines a derived object is shown in the next section.)

Editing of objects and text within objects: To manipulate objects and text within the objects, we use the EXPAND, ERASE, MOVE, SCROLL, LOCATE, ZOOM, and PUSHDOWN function keys.

Our main concern with the system is its ease of use, which we believe is partly related to fewer function keys for more operations. To achieve that, the operations carried out are a function of the position of the cursor at the time the function key is pressed. For example, if one presses the EXPAND function key, the action that occurs depends on the location of the cursor, as follows:

- If the cursor is positioned on the vertical boundary of a WINDOW, the window expands vertically.
- If the cursor is positioned on the horizontal boundary of a WINDOW, the window expands horizontally.
- If the cursor is positioned on a vertical boundary of an object within a window, on a letter, for example, the letter expands vertically.
- If the cursor is positioned between two words in text within an object, a space appears between these words for insertion of additional text, and the rest of the text automatically wraps around to adjust for that space.

This paradigm follows for the other function keys. If one places the cursor on an object name, e.g., a table name, and presses the MOVE function key, the system marks that position with an indicator. Then when one moves the cursor to any other position and presses MOVE again, the object moves from the marked position to the new cursor position. Similarly, one can move text from within an object by first marking the beginning and the ending of the text string, and then moving the cursor to the new position where the text is to be imbedded. In this way one can achieve the cut-and-paste operation that exists in conventional word processors, using the same function key that moves two-dimensional objects. The scope of each function key (which depends on the cursor position) starts with the editing of two-dimensional windows and descends hierarchically to objects within windows, to fields within objects, and ends with text within fields.

Briefly summarized, the above function keys perform the following operations.

EXPAND — Expands windows and objects vertically and horizontally, expands fields in tables, adds new columns or rows to a table or a report, expands text by creating space at the cursor position.

ERASE — Erases windows and objects within windows, erases rows and columns of tables and data in fields of objects, shrinks windows and objects horizontally or vertically depending on cursor position, and erases text within an object if the beginning and end of the desired deletion are marked.

MOVE — Moves windows, objects within windows, and text within an object by marking beginning and end of the text to be moved. The text is imbedded at the new location, whether in the same object or in a different object of a different window.

SCROLL — Scrolls all windows and their objects if the cursor position is in the space between windows; scrolls all the objects within one window if the cursor position is on a particular window; and scrolls text within an object if the cursor is within the object. For example, if the output of a query requires more lines than the table has, one can scroll the output records with that space.

LOCATE — Locates windows, objects within windows, or text within the objects by specifying what string of characters one wants to locate. For example, if cursor position is on a letter object and one presses the LOCATE function key, the system asks one to enter the string he wishes to locate. The system then highlights occurrences of that string.

ZOOM — When windows or objects within windows are moved or scrolled out of the screen, it becomes difficult to keep track of their location. Therefore, a ZOOM function is provided to scale down all the objects so that they can fit simultaneously on the screen. The scale varies depending on the distance between objects. When scaled down (in a ZOOM mode), a green rectangle is superimposed on the objects as an indication of the screen position if one leaves the ZOOM mode. Reduced objects in a ZOOM mode can be moved by the MOVE function key. In addition, the green rectangle can be moved to capture different objects. When one leaves the ZOOM mode, the object covered with the green rectangle expands to a full-size screen. ZOOM is illustrated in Figure 5.

PUSHDOWN — As previously stated, each object resides on a different depth plane. The PUSHDOWN key allows one to push down the objects pointed to by the cursor to the bottom of the pile of object planes.

This short summary of Screen Manager facilities does not exhaust the list of its capabilities. The only limitation on the number and sizes of objects that one can display simultaneously is that of the virtual

Figure 2 Display of various Office-by-Example objects

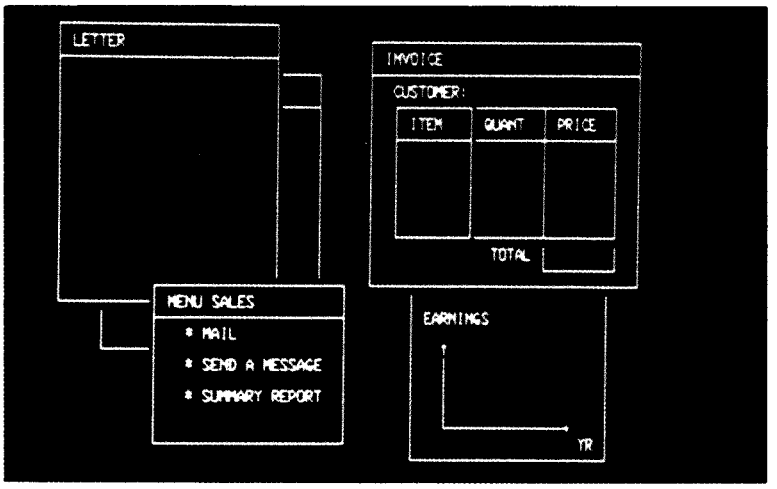
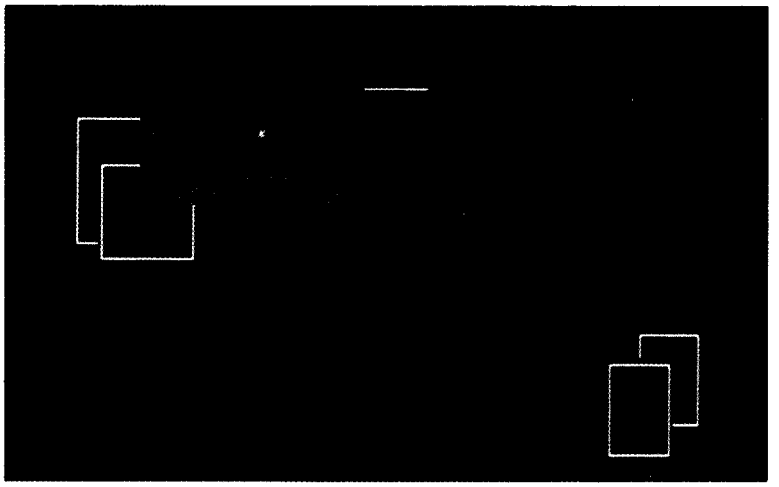


Figure 5 ZOOM mode



address space of the machine. If too many objects clutter a single window, one can use the LOCATE function key to locate each object separately in sequential order.

word processor

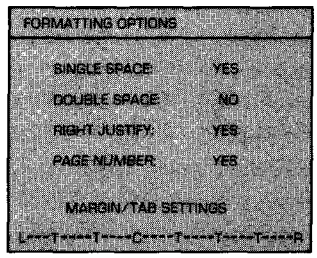
As we mentioned in the Screen Manager section, most of the text editing operations are accomplished by the Screen Manager (ERASE, EXPAND, LOCATE, MOVE, etc.). If one wants to have an easy-to-use system, the style of text editing must be the same as that of editing two-dimensional objects. With a system architecture that can support a local processor, text editing and formatting can be carried out locally while data processing and other operations can be accom-

plished in the host processor. (Note that these considerations are transparent to the user, who is not and should not be concerned with where each part of the program is executed.)

formatting

At present, the Screen Manager is responsible for formatting the objects and their texts. (Later on, this function should become part of the word processor.) The way this is done is somewhat similar to the present Displaywriter implementation. That is, when the user creates a document, he may call a FORMATTING OPTIONS box to be superimposed on top of that document. This box has various formatting options with their default values as shown in Figure 6.

Figure 6 Formatting options for objects and their texts



The user may change these defaults permanently by updating the box, or temporarily by changing a YES to a NO option or vice versa. The option values are also cursor dependent, and they remain in effect until the next change in options. Furthermore, formatting is carried out interactively, and each time the ENTER key is pressed the document is reformatted. The width of the FORMATTING OPTIONS box is the same as that of the object being formatted. Thus the displayed ruler can be used to set up tabs and margins as shown in Figure 6.

data base definition processor

As in QBE, OBE users may define their own data bases. Data bases can be defined either as collections of relations (tables) or as hierarchical views of relations. This is accomplished by displaying a blank skeleton of a table (in the case of relations) on the screen and filling in the table name, column names, and field attributes (such as TYPE, IMAGE, KEY, etc.). An example of such a definition is shown in the next section. In the case of a hierarchy, the user displays a blank skeleton of a hierarchy on the screen (which consists of tables pointing to other tables in a hierarchical fashion) and then fills the blanks in the same manner as the relations.

forms and reports

A *form* in OBE is a generalization of a table and is viewed as a data object. There is a major difference, however, in that a table (relation) is a Screen Manager primitive object (i.e., its skeleton structure is stored and can be displayed by pressing a function key), whereas a form skeleton is a derived object and must first be constructed by the users before defining its various fields. Once the skeleton form is constructed, the user links its fields to fields in columns of an associated table(s), and defines these fields almost as though they were table fields. A form is considered to be a data base object and thus can capture data, and its data fields can be modified. One can also issue a query against a form. (An example of form definition is shown later in this paper.)

Fields of a form are mapped into fields of corresponding table(s) so that there is no need to define a special query language for forms. Thus the same syntax used to express queries or modifications in fields of relational table(s) can be used for fields in a form. This is done because internally the form fields are mapped into table fields.

Also, integrity and authority constraints can be specified on the form fields, again to save the definition of a new form syntax. An example of a form definition is shown later in this paper.

A *report,* on the other hand, is an output object only. It can, of course, be stored globally as a string of text, but its structure is not stored in the system. Thus data cannot be captured via a report, and one cannot query a report.

As we shall see in the examples later in this paper, we distinguish between two types of reports: *single-stage report* and *two-stage report.* In a single-stage report, all fields are filled in by data copied from a data base. In a two-stage report, some of the fields are first copied from a data base. In the second stage, further processing is carried out on the report itself to calculate other fields. For example, totals and subtotals may be calculated after the data are copied from the data base to the body of the report, or in the case of reports in which one wants to total all the fields horizontally.

query translator and processor

After the Screen Manager passes the linear string representing the various objects and their entries, the string is passed to the Query Translator, which parses it and checks the entries against the syntax grammar of the various fields. At this point, conflicting data TYPEs, invalid key words, etc. are detected and appropriate error messages are passed back to the user. For example, if one links a character field TYPE to a numeric field TYPE, an error message is issued to indicate that there are incongruent fields. After the program passes the various checks, an internal representation of the query is created and is passed to the Query Processor (QP) for selecting efficient search criteria for the relations. The Query Processor, in turn, issues calls to the HRMS (to be explained later) to perform such operations as create relations, scan relations on particular selection criteria, create inversions (indices) on certain columns of a relation, and sort relations on particular columns.

trigger manager

A requirement of fundamental importance to office and business automation is the ability of the system to act automatically when it detects specified conditions. This allows the user to automate many routine business procedures so as to devote more time to nonroutine tasks. The following is a list of actions one may want to automate:

- Deferred messages, such as prestored messages or reports to be sent at predetermined dates or frequencies.
- Objects to be automatically sent if a condition is met in the data base, such as prestored messages to be sent automatically to managers who exceed their travel budgets.
- Follow-up procedures, such as an alert message automatically sent to a user if a piece of outside correspondence is not answered within a given time.
- Acknowledgments.

- Update of the data base, such as automatic budget increases of individual managers by an amount indexed to overall program budget increases.
- Creation of logs, such as the insertion of the name and address on every outgoing letter into a predefined mail log table.
- Inventory replenishment, such as the automatic sending of a reorder message when the quantity of a stock item falls below a certain level.

Most of the procedures just given can be automated by *trigger expressions*. A trigger expression is defined as a labeled QBE expression that activates either an action or another trigger expression when specified conditions are met in the data bases. A trigger expression is evaluated either upon modifications of the data base or on the basis of time (i.e., at specified times and dates or at specified intervals).

One of the tasks of the trigger manager is to keep and maintain directories of all the trigger programs in the system. In the case of time triggers, i.e., when trigger programs have to be executed either at specified intervals or at specified future times, the trigger manager, before logging off, passes to the underlying operating system the date and time at which the OBE system must be awakened to evaluate the next trigger program. We are assuming here that the underlying operating system has an automatic logon feature such as in CMS. The trigger manager also keeps track of the actions to be carried out if a particular trigger is activated.

menus and stored programs

Two-dimensional programs can be stored in OBE by naming and storing a *window*. A window may contain a command box that calls for the execution of other programs. In this manner, the nesting of programs is accomplished. Another way to execute a collection of programs one at a time is by means of *menus*.

A menu is a box that can be defined by end users. The menu definer can enter a sequence of various program names into a menu. Then if one places the cursor on one of the menu selections and presses the enter key, the underlying program is executed and the results displayed. If, for example, one selects NEW MAIL for execution, an underlying query program is carried out to do a simple query on the incoming mail table.

access to and extraction from IMS data bases

For an office system to be useful, it is essential to provide the office worker some access to large central data bases that in most cases are stored as IMS hierarchical structures. It is the provision of such facilities that determines whether an executive or a principal is going to make the effort to learn a computer system, no matter how simply it has been designed.

Conventionally, the only way to access IMS data bases is by writing a DL/I batch program. OBE, on the other hand, provides an easy way for

an end user to call the IMS hierarchical structure, which is then displayed as pictures on the screen. The user proceeds by filling in the appropriate fields using the desired selection criteria. The system, in turn, translates this high-level nonprocedural program into a conventional DL/I-PL/I program that is to be shipped to IMS for processing. (We are, of course, assuming that the workstation is connected via a proper network to the IMS DB-DC processor.)

Furthermore, in many cases, one wants to extract a data subset from IMS and map it locally into a different storage structure, such as a relational structure. This can also be achieved by the end users by specifying the mapping on the screen (via example elements, as shown later in this paper). The current OBE implementation does not include the capability to update an IMS data base from OBE.

authority subsystem manager

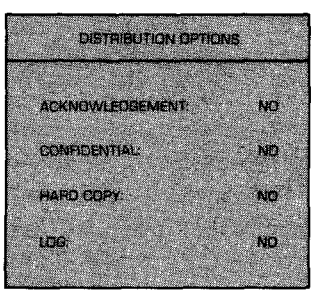
As in QBE, OBE has sophisticated means by which users can specify the delegation of authority. The creator of a table can, for example, specify read, insert, delete, and update options for either an entire table or a subset of a table, as determined by any query expression. The same applies to the creation and modification of documents.

When a user issues a request to access or modify a data base that may contain tables, forms, documents, etc., the Screen Manager passes the request to the authority manager, which checks its validity. If the request is valid, the Screen Manager proceeds to process the request in the normal manner. Otherwise the user is notified that the request is denied for lack of authority.

Communication Manager

If an object of OBE is sent to other nodes of the system, or if an object is received from other nodes, the *Communication Manager* interacts with the underlying operating system to distribute such an object according to user specification. The Communication Manager keeps directories of user IDs of other locations. It also keeps directories of users' names and their IDs and department names and their node IDs, since one can send an object to a user's name, a department name, or both.

Figure 7 Distribution options



When an object is requested for distribution, the Communication Manager displays to the user a DISTRIBUTION OPTIONS object, as shown in Figure 7.

Users have the option to change default values from NO to YES and vice versa. If, for example, the ACK attribute is changed to YES, the Communication Manager sends a request (attached to the object) to the receiver for acknowledging the receipt of the object. If the LOG option is changed to YES, the object is logged before it is sent. If the CONF option is changed to YES, the Communication Manager encrypts the confidential contents of the object and only by entering a special key does the receiver have access to that object.

Received objects are passed from the operating system to the Communication Manager, which logs them in a MAIL table and informs the user about the receipt of the object.

document creation, storage, and retrieval

In OBE, a text document may be thought of as a report, that is, as a report containing only text without any formal structure. Therefore, a document can be stored in the same way as a report and other objects.

A document can be retrieved by its name (as are tables, forms, reports, etc.), but it can also be retrieved by the text content. Since OBE is a superset of QBE, we are using the same partial-example-element feature for searching by a word or sentence within the body of the text. Furthermore, we use the same aggregate functions to count documents that satisfy search criteria. For example, one can easily request a count of all documents that have the words "computer applications" in their texts. Similarly, one can request those documents to be displayed on the screen for observation.

Hierarchical Relational Memory Structure (HRMS)

The OBE system interface to the operating system file manager is called the *Hierarchical Relational Memory Structure (HRMS)*. HRMS provides the rest of the system facilities with a relational view of the data base. Thus, one can issue calls to HRMS to perform such operations as:

CREATE a relation
SCAN a relation
UPDATE a relation
CREATE INVERSIONS (index)
DROP INVERSIONS

The HRMS interface to the system can be viewed as a relational access method interface, such as XRAM, which was used in the QBE product. The one major difference is that it does not carry out any disk accesses. We are assuming that a relatively large virtual memory is available to a single user. This assumption seems reasonable for the following reasons. A local workstation is not likely to be shared by many users. Memory cost is decreasing and so memory is becoming more available. We are assuming relatively small data bases, so that they fit in the computer's virtual address space.

Initial testing of the HRMS performance indicates significant improvement over the performance of the Query-by-Example product access method. Also, for medium-size data bases of several thousands of records, the resources required are significantly reduced.

Examples of OBE programs

Although the OBE language is very rich in facilities, its syntax and concepts are simple. To begin, a new user needs to learn very little more than the following:

1. Operators on programs or data within objects.

P.	to present or display
I.	to insert
D.	to delete
U.	to update
S. or SEND.	to send
G.	to group

2. Example elements—underlined string of characters.
3. Aggregate functions.

SUM.	to sum or total multiple values
CNT.	to count multiple values
AVG.	to average multiple values
MAX.	to find the maximum of multiple values
MIN.	to find the minimum of multiple values
UNQ.	to eliminate identical values

As in the case of the Screen Manager function keys, the OBE operators work in a consistent manner and are context dependent when placed in different fields within the objects. For example, if the P. operator is placed on a window heading, it retrieves the names of all stored windows. If P. is placed on a table heading, it displays the names of all the stored tables in the system. If, on the other hand, P. is placed within a field of a table, it displays the data associated with that field. The same holds true for the rest of the operators: I., D., U., S., and G.

examples of OBE programming

We now illustrate OBE programming with some examples.

Document creation, storage, and distribution. To create a document(s), one displays its skeleton on the screen and proceeds by entering the text. The text can be edited by the Screen Manager function keys as previously explained, and formatted dynamically using the formatting options of Figure 6. Since objects can be moved in different planes, one can create more than one object simultaneously. After the documents are created, one can store them by placing an I. in front of their names. If one wants to distribute them to other users, an S. command is placed in the OBE command box. These operations are shown in Figure 8. Here, the I. command stores the documents, and the first S. command sends the letter to two recipients. Before the system distributes these documents, a distribution option shown in Figure 7 is displayed for each document, and one has the choice of changing the options (YES or NO) as explained. Thus the user need not remember any of either the formatting options or the distribution options because they are automatically displayed on the screen. All one has to remember is to change the option YES to NO or vice versa.

Figure 8 Composing, storing, and distributing two documents

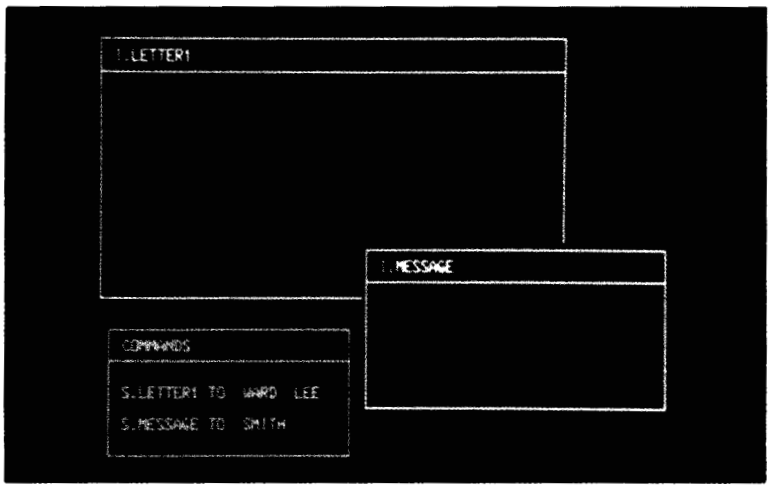
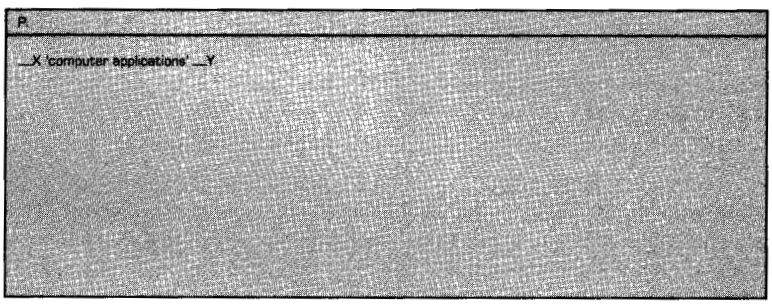
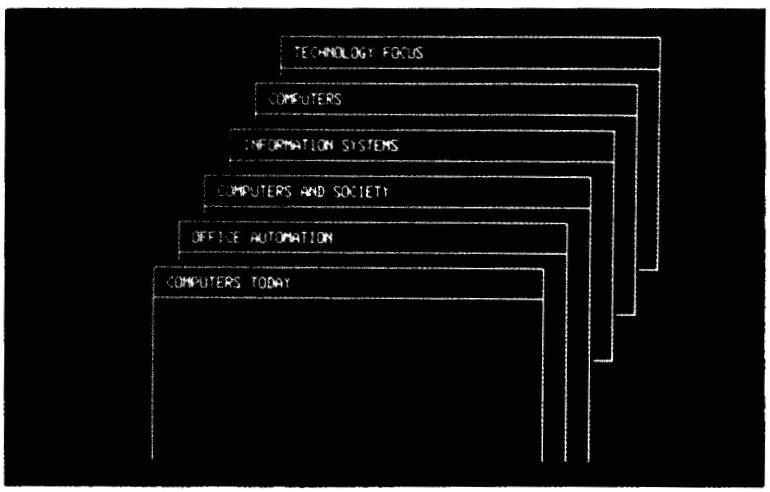


Figure 9 A program (A) to retrieve documents containing the words "computer applications," and (B) the retrieved documents



(A)



(B)

Document retrieval, deletion, or update. Stored OBE documents can be retrieved by their names by placing a P. followed by the document name in the document name field. Thus P.LETTER1 in the heading of a document box retrieves the contents of the document LETTER1.

Another way of retrieving documents is by using search criteria on the text. This is achieved by specifying the search criteria in the body of the text, as shown in Figure 9(A). The program retrieves all documents that contain the words "computer applications" in their bodies (Figure 9 (B)). If, for some reason, one wants to count the documents, the operator and function P.CNT. is used instead of P.. This is an instruction to display the number of documents that were found to meet the specified search criteria. This, of course, is a simple example. Because of the richness of OBE, far more complex search criteria can be used than the one illustrated here, by entering more example elements and specifying more logical conditions in the condition box.

Documents are deleted (destroyed) by entering a D. followed by the document name, or updated by entering U. followed by the document name.

Data base query and modification. Consider a relational data base consisting of two tables: SALES, with the column headings SALESMAN, SALES QUOTA, and SALES TO DATE; and ITEM SALES, with the headings ITEM, PRICE, SALESMAN. We start with a simple query on the SALES table. (We are assuming the tables have been defined.)

Simple table selection. List, for example, the names of salesmen who have exceeded $50 000 in sales. Initially, the user displays a blank table skeleton. He then enters the table name into the table name field (in this case SALES). The system then generates the column heading automatically. Having established the column headings, the user then programs within the skeleton by making the entries shown in Figure 10. P. stands for either print or display. It indicates that the desired outputs are names of salesmen and sales-to-date amounts, but only of those who exceed $50 000.

Cross-referencing between fields. Suppose a manager wants to find the names and telephone numbers of salesmen who exceeded $50 000 in sales, and suppose further that there is a DIRECTORY table that contains the employees' NAME and PHONE #. This query is accomplished by displaying two table skeletons on the screen and entering both table names in the appropriate spaces. The final formulation of this query is shown in Figure 11.

The example element (variable) N in both tables causes the listings in the NAME field to match the SALESMAN field; that is, only names and telephone numbers of salesmen who made over $50 000 in sales are to be displayed.

Figure 10 Simple table section

SALES	SALESMAN	SALES QUOTA	SALES TO DATE
	P.		P. > 50000

Figure 11 Cross referencing between tables

DIRECTORY	NAME	PHONE #
	P. _N	P.

SALES	SALESMAN	SALES QUOTA	SALES TO DATE
	_N		> 50000

Figure 12 Cross referencing in the same table

SALES	SALESMAN	SALES QUOTA	SALES TO DATE
	P. SMITH		> _S _S

Cross-referencing can also be done within records of the same table. Consider, for example, the query to "List the name(s) of salesmen whose sales to date exceed that of Smith." This query is shown in Figure 12. The order of the rows is immaterial.

Conditions on field values. List the names and sales to date of salesmen who have exceeded their sales quotas. This is shown in Figure 13. A condition box primitive object can be displayed through a function key for the specification of conditions.

Figure 13 Using a condition box

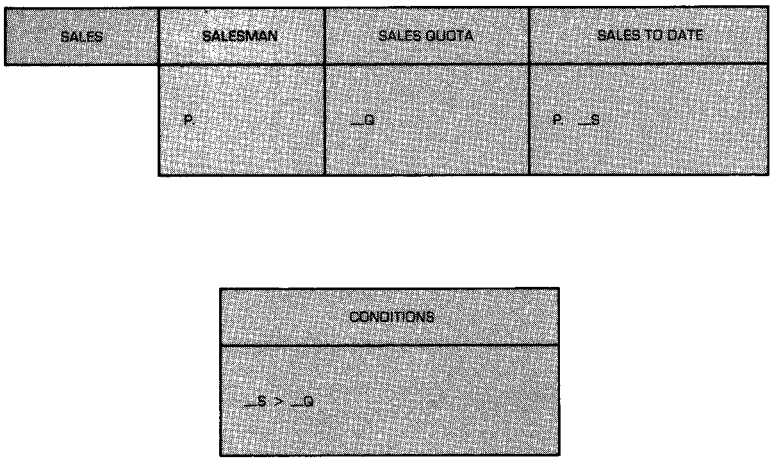


Figure 14 Simple retrieval from an IMS data base

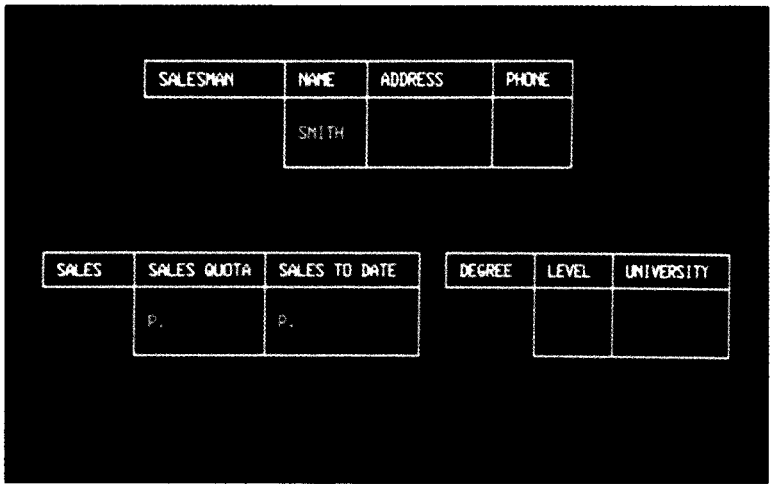
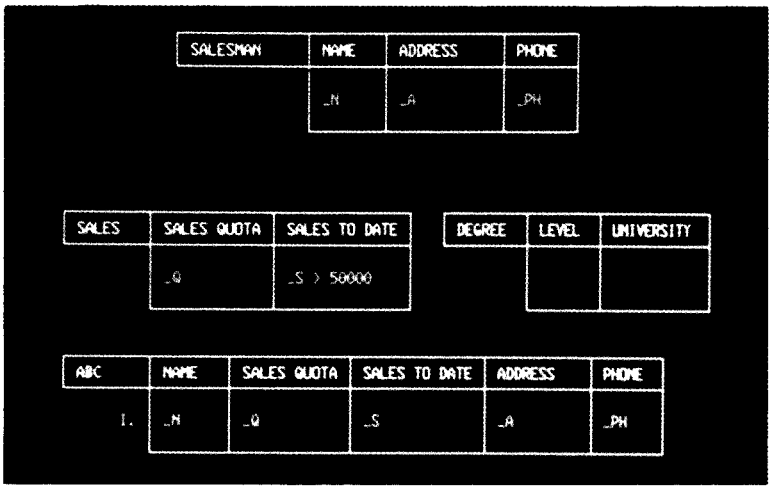


Figure 15 Extraction from IMS and copying into a relational data base



To modify the data base, the operator's I., D., and U. are used accordingly. Further examples of data base query and modification are given in References 1–7.

Retrieval from IMS data base. In the case of IMS data bases, the user displays a blank skeleton of a hierarchy on the screen and proceeds by entering the IMS hierarchy name (Program Communication Blocks, PCBs) in the appropriate space. The system then automatically displays the hierarchical segments, which the user may access, and their headings. This information is mapped from the IMS Data Base Descriptions (DBDs) and PCBs directories.

Having established the hierarchical structure on the screen, the user fills in the spaces by the OBE operators, as shown in Figure 14.

IMS extraction. By using example elements, one can extract data from IMS and copy them into a relational data base. This is done by displaying the hierarchy and the relation(s) skeletons and then inserting (I.) example elements into the relation(s) linked to fields in the hierarchy, as shown in Figure 15. For further details on hierarchical data base structures, see Reference 7.

Distributing merged text and data objects. In the previous examples, we have illustrated word processing and data base operations. We now combine the two. Suppose a manager wants to send congratulatory letters to all salesmen who have exceeded their sales quotas. This is easily accomplished in OBE by mapping appropriate example elements from the data base to the body of the document, as shown in Figure 16.

Note that the S. to N in the command box establishes a dynamic distribution list that is determined from a data base query. This program, therefore, sends personalized letters to all salesmen who have exceeded their quotas.

The example illustrated in Figure 16 is seemingly a rather trivial one. One can imagine far more complex applications involving lengthy documents or reports with example elements linked to more than one data base table or IMS hierarchy and with more than one condition in the data base. The purpose of this example is to demonstrate the elegance of combining the activities of word processing, data processing, and communications into a single unified interface.

Graph composition. Example element mapping can also be used to generate graphs. Consider a COMPANY table that lists the earnings of various companies by calendar year under the headings of NAME, EARNINGS, and YEAR.

The formulation for a graph for each company that contains its year-by-year earnings is shown in Figure 17. Here each company

Figure 16 Distribution of personalized letters

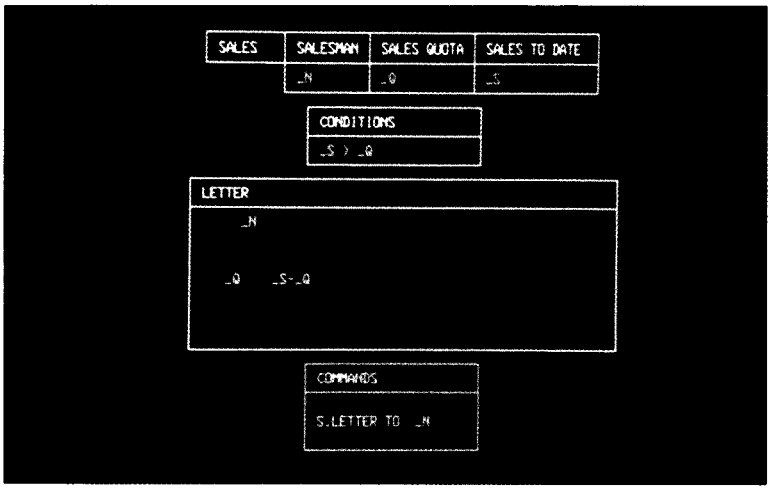
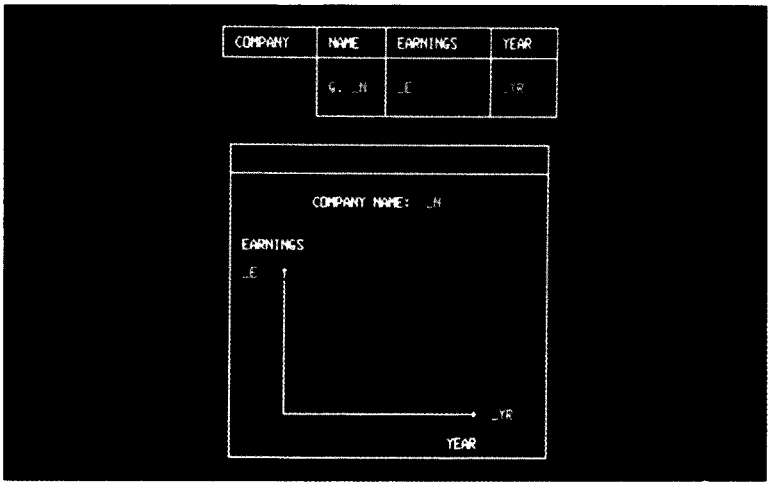


Figure 17 Example element mapping used to create a graph



name is mapped to a separate report. The x-y coordinates are called YEAR and EARNINGS, respectively, and their values are mapped via the example elements YR and E. The system adjusts the scale of the coordinates to take full advantage of available space.

Table definition. We now give an example of how end users define (create) tables in OBE. Starting from a blank table skeleton, the user inserts the table name and the column headings as shown in Figure 18. When processing the entries in Figure 18, the system displays to the user the row attributes shown in Figure 19. The user then proceeds to fill in the attribute values as shown in Figure 20, as follows:

Figure 18 Creation of table headings

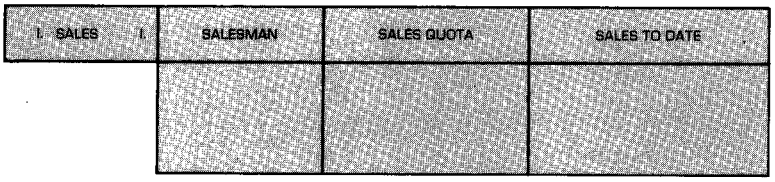


Figure 19 Display of row attributes

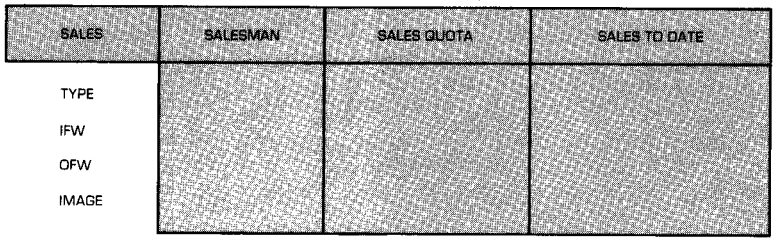


Figure 20 Definition of row attributes

SALES	SALESMAN	SALES QUOTA	SALES TO DATE
TYPE I.	CHAR	FIXED	FIXED
IFW I.	15	12	12
OFW I.	--	--	--
IMAGE I.		--------#.##	--------#.##

- TYPE specifies the field type: CHAR, FIXED, FLOAT, DATE, TIME.
- IFW (Input Field Width) specifies the desired width of the field when displayed on the screen for input.
- OFW (Output Field Width) specifies the maximum desired width of the field when output is displayed on the screen.
- IMAGE specifies the image one wants the output to have. For example, in Figure 20, the amounts have two characters after the decimal points. (IMAGE does not refer to CHAR data type.)

Form and report definition. Forms and reports are defined in the same style as tables. In this case, however, the user first graphically structures the image of the form or report on the screen, mimicking manual construction. The user then enters example elements in the

Figure 21 Constructing the structure and headings for a form

I. INVOICE

CUSTOMER: _X

ITEM	QUANTITY	UNIT PRICE	EXTENDED
_I	_Q	_UP	_EX
		TOTAL	

Figure 22 Form field definition

TABLE	INVOICE	CUSTOMER	ITEM	QUANTITY	UNIT PRICE	EXTENDED
		_X	_I	_Q	_UP	_EX
TYPE	I.	CHAR	CHAR	FIXED	FIXED	FIXED
FW	I.	15	10	8	10	12
FLINE	I.	YES	NO	NO	NO	NO
PB	I.	YES	NO	NO	NO	NO
SV	I.	NO	NO	NO	NO	NO
OP	I.	YES	YES	NO	NO	NO

fields to be defined. As an example form, an invoice has been constructed and headings given in Figure 21.

As in the example of the table, when the user processes the entries in the form in Figure 21, the system prompts the user to provide data attributes by displaying a corresponding table skeleton with the same name, headings, and example elements as in the form. This table structure includes row attributes for the definition of the report fields, as shown in Figure 22.

The user proceeds by entering values for the various row attributes, which are defined as follows:

TYPE and FW — As in the table definition for Figure 20.
FLINE (Fold Line) — YES means that if the output data string is larger than the FW, then fold the string on the second line.
PB (Page Break) — YES means that every time the value of the field is changed, a new page is started.
SV (Single Value) — YES means that the corresponding example element cannot have more than one value.
OP (One Print) — YES means to print only one of identical values.

Also note that the number of row attributes is open-ended, so that other row attributes can be added as required to implement future system capabilities.

Figure 23 MATRIX report structure

	JAN	FEB	MAR	TOTALS
HENRY				
LEE				
SMITH				
TOTALS				

After the definition of the form is complete, one can copy data from other tables into that structure, again by means of example elements.

Two-stage report. As mentioned earlier, it is sometimes desirable first to move some of the data from the data base tables and then operate locally on the data of the report to produce totals and subtotals, etc. To demonstrate this, assume that we constructed a MATRIX report to summarize the quarterly sales of three salesmen, as shown in Figure 23. The program that copies data from a data base table(s) to the body of the report is shown in Figure 24. Note that the example elements correspond to Single Value (SV) data moved from SALES1 table. The SUM. functions act to total these values both vertically and horizontally. This is a two-stage report because the sums can be carried out only after the data are placed in the report structure.

Figure 24 Filling in a MATRIX report

SALES1	SALESMAN	SALES	MONTH
	HENRY	_S1	1
	HENRY	_S2	2
	HENRY	_S3	3
	LEE	_L1	1
	LEE	_L2	2
	LEE	_L3	3
	SMITH	_N1	1
	SMITH	_N2	2
	SMITH	_N3	3

	JAN	FEB	MAR	TOTALS
HENRY	_S1	_S2	_S3	SUM.
LEE	_L1	_L2	_L3	SUM.
SMITH	_N1	_N2	_N3	SUM.
TOTALS	SUM.	SUM.	SUM.	SUM.

Receiving objects. To keep track of received objects, the system maintains a log called MAIL, which contains the headings shown in Figure 25. When an object is received from other users, the system automatically enters a record in the MAIL table that indicates from whom and to whom the object was sent, as well as the object name, classification, etc. The MAIL table is accessible to all users, but various authority statements restrict access to particular subsets of it. For example, one can issue an authority statement restricting a user to reading his own mail only.

trigger programs

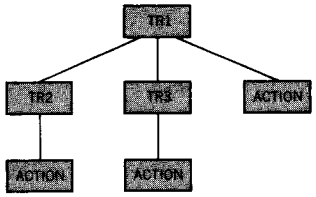
The syntax of a trigger expression consists of the identifier TR followed by a trigger name. (Trigger names are unique within a single user ID.) TR1 and TRABC are examples of valid trigger identifiers. Triggers can have various parameters to indicate the frequency with which corresponding QBE expressions are to be evaluated.

Figure 25 MAIL log to track received objects

MAIL	FROM	TO	OBJECT	DATE	TIME	TYPE	OLD/NEW

There are two types of triggers. *Modification triggers* are evaluated upon a change in the data base. For example, TR1(D.) means "evaluate trigger expression 1 upon deletions," and TR2(I.,U.) means "evaluate trigger expression 2 upon insertions or updates." *Time triggers* are evaluated at a specified time and date or at specified time periods. For example, TR5(5/5/82 AT 16:00) means "evaluate trigger expression 5 on 5/5/82 at 4 pm," and TR6(DAILY) means "evaluate trigger expression 6 daily." Keywords can be used to specify the period: HOURLY, DAILY, WEEKLY, MONTHLY, every WORKING DAY, etc. If a trigger expression has no parameters—such as TR7—it is evaluated when the stored program containing this trigger expression is executed.

Figure 26 Example sequence of trigger actions



A trigger is said to be activated if its corresponding QBE expression is satisfied. As previously mentioned, when a trigger is activated, it causes either an action or the evaluation of another trigger expression. That is, one trigger can be a parameter of another trigger, as shown in Figure 26.

The leaves of the tree in Figure 26 must always represent actions because the activation of a trigger without subsequent effect is meaningless. The following actions can take place upon the activation of triggers:

DISPLAY DATA OBJECTS
INSERT DATA OBJECTS
UPDATE DATA OBJECTS
DELETE DATA OBJECTS
SEND DATA OBJECTS
EXECUTE PROGRAM OBJECTS

The syntax to carry out these actions is a generalization of the P., I., U., D., S., and E. operators with trigger expressions as parameters:

P (TR1 AND/OR TR2 . . .).
I (TR1 AND/OR TR2 . . .).
U (TR1 AND/OR TR2 . . .).
D (TR1 AND/OR TR2 . . .).
SEND (TR1 AND/OR TR2 . . .). or S(TR1 AND/OR TR2 . . .).
EXECUTE (TR1 AND/OR TR2 . . .). or E(TR1 AND/OR TR2 . . .).

The following examples explain the above syntax:

Figure 27 Trigger program to check SALES table and send letters

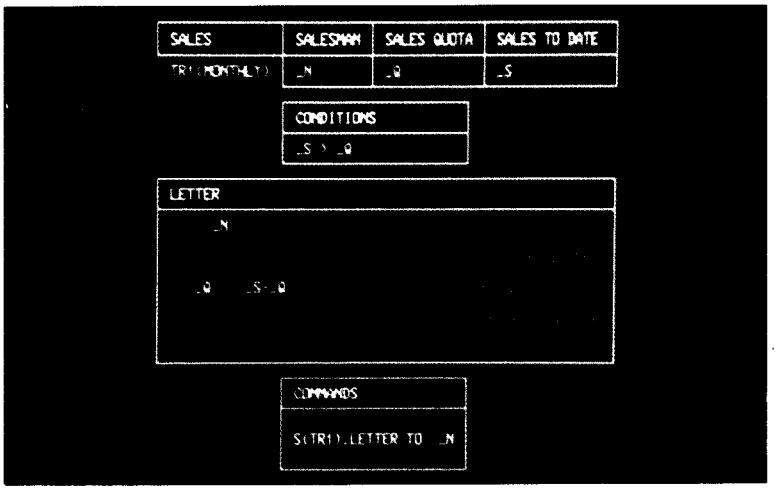
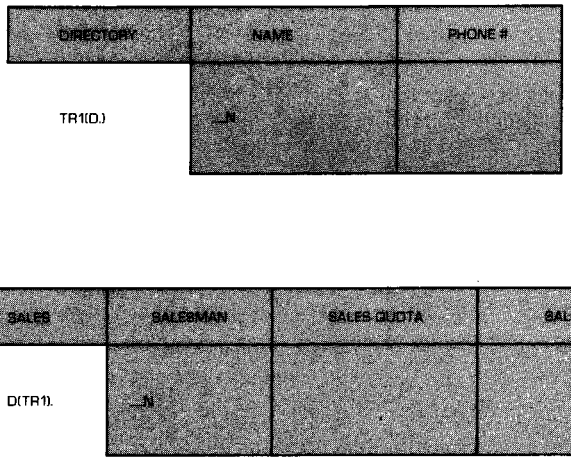


Figure 28 Deletion of a record from the DIRECTORY table triggers the deletion of the corresponding record from the SALES table



I(TR1). Insert records in the corresponding table only if trigger 1 is activated.
SEND (TR5 OR TR6).A Send object A only if either trigger 5 or trigger 6 is activated.
EXECUTE (TR8).ABC Execute program ABC only if trigger 8 is activated.

examples of trigger programs

Consider the program for the distribution of personalized letters shown in Figure 16. Suppose Henry wants the system on a monthly basis, to check and automatically send congratulatory notes to

Figure 29 Monthly sales report program

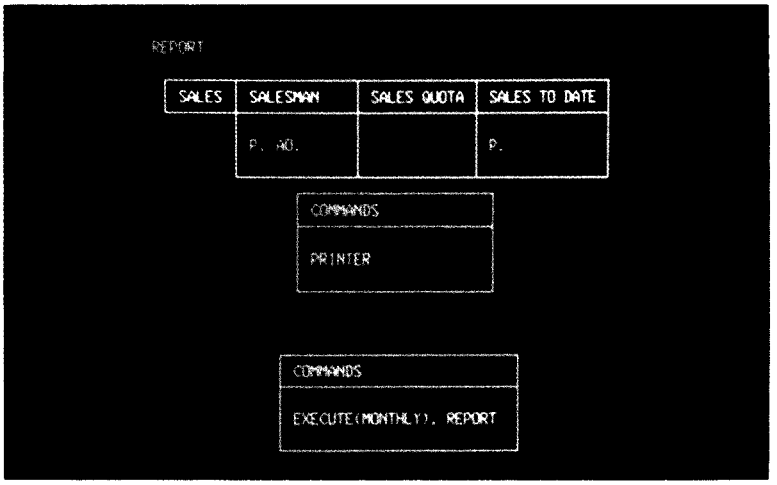
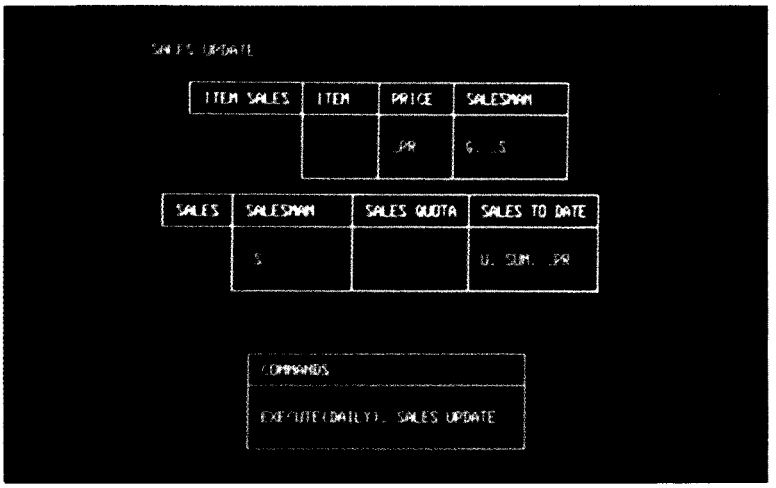


Figure 30 Program for daily sales update



salesmen who have exceeded their quotas. This can be achieved by adding to the program in Figure 16 trigger expressions as shown in Figure 27. Each month the trigger program checks the SALES table and sends letters to salesmen who exceed their quotas.

Another example is the ability to modify one table as a consequence of the modification of another. In doing this, triggers can be used to ensure that referential constraints (18) are not violated. To illustrate this operation, Figure 28 shows the deletion of information from the

SALES table of a salesman who is deleted from the DIRECTORY table. Upon the deletion of a record from the DIRECTORY table, trigger TR1 is activated and, in turn, deletes the corresponding record from the SALES table.

Application development

The OBE features and facilities described in the previous sections are now used to show that one can write an entire application to automate various business and office procedures. Although the following application dealing with sales management is simple, it demonstrates the generality and the flexibility of the language.

Assume, as before, that a certain department store keeps information on items and their prices sold by individual salesmen (for commission purposes) in a table called ITEM SALES, in addition to the previous table SALES, which is the cumulative monthly sales volume. These two tables can be used to write the following high-level programs.

First is a program to issue monthly reports listing salesmen and their total sales, sorted alphabetically by salesman's name. The program illustrated in Figure 29, arbitrarily called REPORT, produces a hard copy listing of names and sales to data. The AO. function sorts the list into alphabetical order and the command PRINTER produces the hard copy. The statement at the bottom of the figure causes the execution of this program on a monthly basis.

A program for updating the SALES TO DATE column of the SALES table on a daily basis to reflect the daily sales is shown in Figure 30. Here, the program SALES UPDATE sums for each salesman the prices of all items sold and updates the SALES TO DATE field accordingly. The second statement executes the program on a daily basis.

A trigger program shown in Figure 31 checks the SALES table on a monthly basis and sends a report to the manager, Henry, only for salesmen who exceed their quotas. The report should include the salesmen's names, their individual sales amounts, count of number of salesmen exceeding quota (CNT.N), and the total of all sales (SUM.S).

These three programs illustrate the expressive power of OBE. Of course, more complex programs involving many more tables, conditions, triggers, messages, and reports can be composed.

The application illustrated in Figure 32 is that of the production and distribution of invoices from a data base of fulfilled orders and item prices. This program produces an INVOICE for each customer that lists items, quantities, prices, and totals. The S. command in the box causes the distribution of the invoices. In our terminology, the INVOICE is considered to be a two-stage report.

Figure 31 Sales report program listing salesmen who exceed monthly quota, number who exceed quota, and the sum of their sales

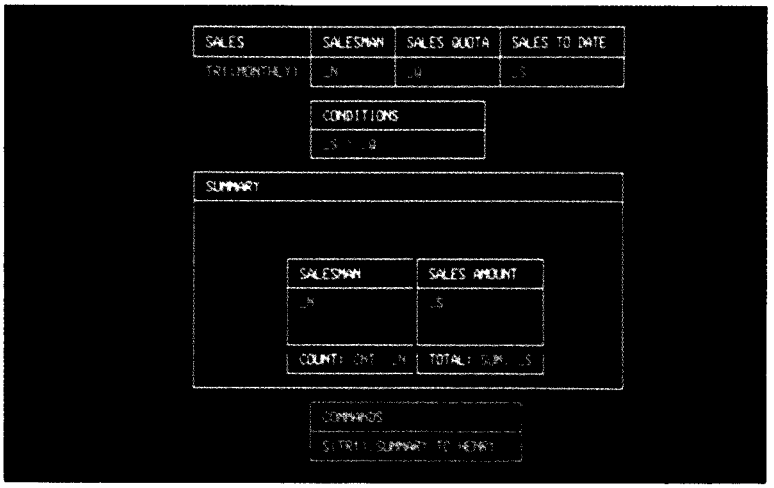
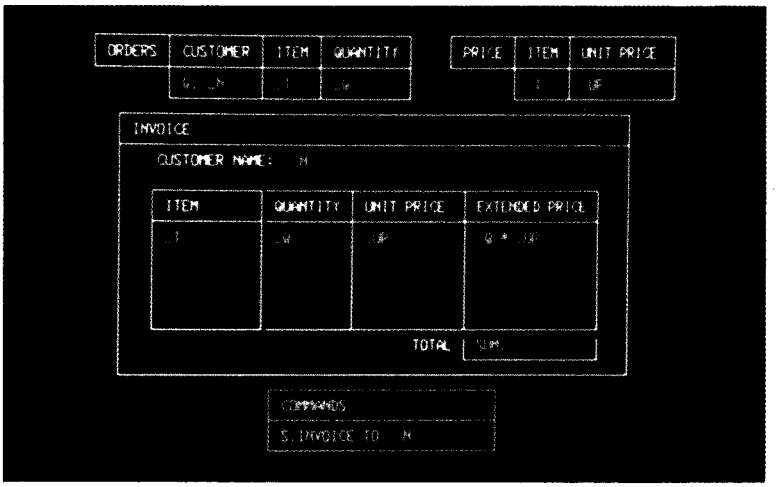


Figure 32 Program to produce an INVOICE for each customer



Menus

As mentioned earlier, menus can be set up by end users for end users. Here is an example of the setting up of a menu by a secretary for an executive according to given specifications. Suppose the secretary wants to create a menu with the following selection options: MAIL, SEND A MESSAGE, and SALES REPORT. As shown in Figure 33, the secretary starts by inserting the key word MENU followed by the menu name in a box, and then proceeds by entering the selections

separated by asterisks. These selections are actually stored program names that the secretary stores in the ordinary manner.

When the executive displays MENU ABC on the screen, places the cursor on one of the selections, and hits the enter key, the system executes the appropriate program and displays the results. Since the execution of a program can cause the display of other objects (in this example, other menus) the users can thus obtain a hierarchy of menus, as illustrated in Figure 34.

For example, if one selects the MAIL option, the result could be another menu, as shown in Figure 35.

If the user now proceeds to select the option NEW MAIL from MAIL menu, the system executes a query program on the MAIL table with the selection criteria NEW under the NEW/OLD field.

In summary, users can easily customize a variety of menus for themselves or others, so that one need not know more about the system than to point at menu selections that have been designed especially for them.

Figure 33 Creating a menu

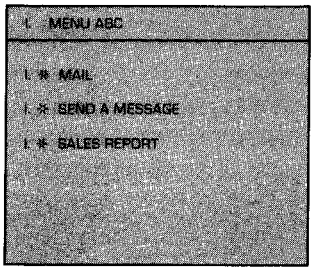
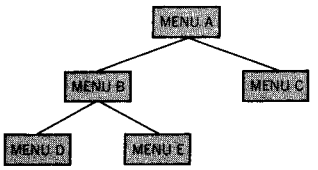


Figure 34 Hierarchy of menus



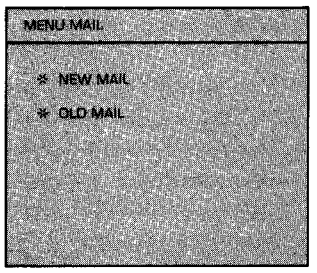
Concluding remarks

In conclusion, the QBE/OBE language is a high-level nonprocedural language with the following unique features:

- Requires very few concepts to get started.
- Gives the feeling of manual manipulation of the objects.
- Combines word processing, data processing, and communication in an elegant, compact, and unified manner.
- Allows users to customize their own menus.
- Allows user access to centralized IMS data bases.
- Allows end users to automate routine business procedures by means of triggers.

ACKNOWLEDGMENTS

I am very much indebted to the Office Automation group for their design and implementation of the various components of the OBE system. In particular, I am grateful to Guy T. Hochgesang for the design and implementation of the Screen Manager and to Arthur C. Ammann for the design and implementation of the HRMS. My thanks also go to Stephen P. Morgan, Ravindran Krishnamurthy, and Erdwin C. Chua for their work on various design aspects of the system. I also want to thank my wife, Rosy, and Erdwin C. Chua for their editorial assistance, and Arthur J. Stein and Erdwin C. Chua for their assistance with the graphics in this paper.



CITED REFERENCES

1. M. M. Zloof, "Query-by-Example: A data base language," *IBM Systems Journal* **16,** No. 4, 324–343 (1977).

2. M. M. Zloof, "Query-by-Example," *AFIPS Conference Proceedings, National Computer Conference* **44,** 431–438 (1975).
3. M. M. Zloof, "Query-by-Example: The invocation and definition of tables and forms," *Proceedings of the International Conference on Very Large Data Bases,* September 1975, 1–24 (1975).
4. M. M. Zloof, *Query-by-Example: Operations on the Transitive Closure,* Research Report RC-5526, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (1975).
5. M. M. Zloof, *Security and Integrity Within the Query-by-Example Data Base Management Language,* Research Report RC-6982, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (1978).
6. *The QBE Terminal User's Guide,* SH20–2078; available through IBM branch offices.
7. M. M. Zloof, "Query-by-Example: Operation on hierarchical data bases," *AFIPS Conference Proceedings, National Computer Conference* **45,** 845–853 (1976).
8. M. M. Zloof and S. P. deJong, "The System for Business Automation (SBA)," *Communications of the ACM* **20,** No. 6, 385–396 (1977).
9. S. P. deJong and M. M. Zloof, *Communication within the System for Business Automation,* Research Report RC-6788, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (1976).
10. D. Tsichritzis, "OFS, An integrated form management system," *International Conference on Very Large Data Bases* (6th), 161–166 (1980).
11. H. L. Morgan, "The future of office of the future," *Office Automation Conference Digest,* AFIPS Press, Roslyn, VA (1980).
12. M. Hammer, *Laboratory for Computer Science Progress Report: Office Automation Group,* MIT Laboratory for Computer Science, Cambridge, MA 02139 (1979).
13. C. Ellis and G. Nutt, *Computer Science and Office Information Systems,* Report SSL-79-6, Xerox Palo Alto Research Center, Palo Alto, CA 94304 (June 1979).
14. D. D. Chamberlin, M. M. Astrahan, K. P. Eswaran, P. P. Griffiths, R. A. Lorie, J. W. Mehl, P. Reisner, and B. W. Wade, "SEQUEL 2: A unified approach to data definition, manipulation, and control," *IBM Journal of Research and Development* **20,** No. 6, 560–575 (November 1976).
15. E. F. Codd, "A data base sublanguage founded on the relational calculus," *Proceedings, ACM SIGFIDET Workshop* (1971).
16. M. M. Stonebraker, *Getting Started in INGRES-A,* Report ERLM 518, Computer Science Department, University of California, Berkeley, CA 94720 (1975).
17. N. C. Shu, V. Y. Lum, S. C. Tung, and C. L. Cha, *Specifications of Forms Processing and Business Procedures for Office Automation,* Research Report RJ-3040, IBM Research Laboratory, San Jose, CA 95193 (January 1981).
18. C. J. Date, *Referential Integrity,* Santa Teresa Laboratory Report TR03.132, IBM Santa Teresa, San Jose, CA 95150 (January 1981).
19. E. F. Codd, "A relational model for large shared data banks," *Communications of the ACM* **13,** No. 6, 377–387 (1970).
20. *Merchandising Management Using Query-by-Example,* CK20-1298-0 (1980); available through IBM branch offices.
21. *Personnel, Finance, and Sales Applications Using Query-by-Example,* CK20-1342-0 (1980); available through IBM branch offices.
22. J. C. Thomas and J. D. Gould, "A psychological study of Query-by-Example," *AFIPS Conference Proceedings, National Computer Conference* **44,** 439–445 (1975).
23. D. Greenblatt and J. Waxman, "A study of three data base query languages," in *Data Bases, Improving Usability and Responsiveness,* B. Schneiderman (Editor), Academic Press, New York (1978).

The author is located at the IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598.

Reprint Order No. G321-5170.